Définition Formelle — Glushkovizer

Résumé

Ce document constitue la définition formelle des différents types de donnée utilisés tout au long de cette librairie. Dans un premier temps, nous nous concentrerons sur les expressions régulières et les fonctions définies sur celle-ci. Puis dans un second temps, nous nous intéresseront aux automates et à leur divers fonctions définies sur eux. Enfin pour finir, nous parlerons d'automates particuliers, ceux de Glushkov, nous aborderons leurs constructions, ainsi que leurs propriétés.

Table des matières

I Prélude

Pour la compréhension de l'ensemble de ce document, nous avons besoin de plusieurs notions de théorie des langages. C'est donc pourquoi cette partie, nous allons étudier les différentes notions nécessaires. Dans un premier temps, nous allons définir ce qu'est un mot et quelles sont les opérations sur les mots. Enfin dans un second temps, nous définirons ce qu'est un langage et quelles opérations sont munis sur les langages.

1 Les mots

Définition 1.1. Un *alphabet* Σ est un ensemble fini de symbole non vide. Un *mot* est une suite finie de symbole sur un alphabet Σ, le mot composé de zéro symbole est appelé *mot vide* est noté ε.

Example 1.1.1.

$$\Sigma = \{a, b, c, d\}$$
$$w = abbcdda$$

Définition 1.2. On parlera de la *longueur d'un mot* w noté $|w|$ pour désigner le nombre de symboles qui le compose. De même, on notera $|w|_a$ pour parler du nombre de a dans le mot w.

Example 1.2.1.

$$w = abbcdda$$
$$|w| = 7$$
$$|w|_d = 2$$

Définition 1.3. Une des opérations les plus essentiels sur les mots est la *concaténation* de mots. On notera donc la concaténation de deux mots $u = a_1 \cdots a_n$ et $v = b_1 \cdots b_n$ par $u \cdot v$. Qui est ainsi égal à $u \cdot v = a_1 \cdots a_n b_1 \cdots b_n$. On définit l'ensemble des mots sur Σ par Σ^*. On notera que :

- La concaténation est associative $(w \cdot u) \cdot v = w \cdot (u \cdot v)$.
- La concaténation admet un élément neutre $u \cdot \varepsilon = \epsilon \cdot u = u$.

Ce qui implique que l'ensemble Σ^* muni de la concaténation $(\Sigma, \cdot)$ forme un monoïde.

Example 1.3.1.

$$u = abab$$
$$v = cdcd$$
$$u \cdot v = ababcdcd$$

Définition 1.4. Grâce à cette opération sur les mots, on peut définir ce qu'est un *facteur*. Un facteur u d'un mot w est une suite extraite de la suite de lettre qui composent le mot w. Autrement dit u est un sous mot de w si $\exists(v,x) \in (\Sigma^*)^2 \mid w = v \cdot u \cdot x$, de plus :

- On parlera de *préfixe* quand $v = \varepsilon$.
- On parlera de *suffixe* quand $x = \varepsilon$.
- Enfin, on parlera de *facteur propre* quand $v \neq \varepsilon \wedge x \neq \varepsilon$.

On remarquera que ε est : *préfixe*, *suffixe* et *facteur* de tout mot.

Example 1.4.1.

$$w = abbcdda$$
$$u = bcd$$
$$y = abb$$
$$w = ab \cdot u \cdot da$$
u est donc un facteur propre
$$w = y \cdot cdda$$
y est un facteur et un préfixe

Définition 1.5. De plus aussi grâce à la concaténation, nous pouvons définir le *miroir* d'un mot w noté $\overleftarrow{w}$. Qui est ainsi défini récursivement comme ceci :

$$\overleftarrow{\varepsilon} = \varepsilon$$
$$\overleftarrow{a} = a$$
$$\overleftarrow{u \cdot a} = a \cdot \overleftarrow{u}$$
Avec $a \in \Sigma$ et $u \in \Sigma^*$

On remarquera qu'un mot est un palindrome si $u = \overleftarrow{u}$.

Example 1.5.1.

$$w = abbcdda$$
$$\overleftarrow{w} = addcbba$$

2 Les langages

Définition 1.6. Un *langage* L est un ensemble de mot sur un alphabet fini Σ. On appellera *langage vide* le langage ne comportant aucun mot, ainsi défini comme ceci : $L = \varnothing$.

Example 1.6.1.

$$\Sigma = \{a, b, c, d\}$$
$$L_1 = \{a, aa, bc, da, \varepsilon\}$$
$$L_2 = \varnothing$$

Définition 1.7. L'équivalant de la longueur d'un mot sur les langages est donc le *cardinal* du langage. On remarquera qu'un langage n'est pas forcément fini et qu'alors son *cardinal* peut être infini.

Example 1.7.1.

$$\Sigma = \{a, b, c, d\}$$
$$L_1 = \{a, aa, bc, da, \varepsilon\}$$
$$L_2 = \varnothing$$
$$|L_1| = 5$$
$$|L_2| = 0$$

Définition 1.8. L'une des opérations les plus importantes sur les langages est l'*union*. On parlera de l'union de langage noté $L_1 \cup L_2$ et défini comme ceci :

$$L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$$

On notera que l'union est associative, commutative et admet un élément neutre ($\varnothing$).

Définition 1.9. À l'image de l'union de langages, nous avons aussi l'*intersection* de langage noté $L_1 \cap L_2$ et donc défini comme cela :

$$L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$$

On notera qu'elle aussi est associative, commutative et admet aussi un élément neutre (Σ^*). On remarquera que $\varnothing$ est un élément absorbant.

Example 1.9.1.

$$\Sigma = \{a, b, c, d\}$$
$$L_1 = \{\varepsilon, a, aa, bc, da\}$$
$$L_2 = \{d, aa, cd\}$$
$$L_1 \cup L_2 = \{\varepsilon, a, d, aa, cd, bc, da\}$$
$$L_1 \cap L_2 = \{aa\}$$

Définition 1.10. Un autre opération importante sur les langages est la *concaténation*. Elle est bien sûr définie en utilisant la concaténation des mots qui compose les langages. Cette opération est ainsi définie comme ceci :

$$L_1 \cdot L_2 = \{u \cdot v \mid u \in L_1, v \in L_2\}$$

On remarquera qu'elle est associative, pas commutative et admet un élément neutre ($\{\varepsilon\}$). Et que $\varnothing$ est aussi un élément absorbant pour cette opération.

Example 1.10.1.

$$\Sigma = \{a, b, c, d\}$$
$$L_1 = \{\varepsilon, a, aa\}$$
$$L_2 = \{d, cc\}$$
$$L_1 \cdot L_2 = \{d, ad, aad, cc, acc, aacc\}$$

Définition 1.11. Par extension, on définit la *copie n-ième* d'un langage L notée L^n et défini récursivement comme ceci :

$$L^0 = \{\varepsilon\}$$
$$L^n = L^{n-1} \cdot L$$

On remarquera que $\varnothing^0 = \{\varepsilon\}$.

Example 1.11.1.

$$\Sigma = \{a, b\}$$
$$L = \{\varepsilon, a\}$$
$$L^3 = \{\varepsilon, a, aa, aaa\}$$

Définition 1.12. Grâce à cette opération, on peut définir l'*étoile* d'un langage notée L^*. Qui peut être défini comme ceci :

$$L^* = \bigcup_{i \geq 0} L^i$$

Example 1.12.1.

$$L = \{a\}$$
$$L^* = \{\varepsilon\} \cup \{a\} \cup \{aa\} \cup \cdots$$

3 Conclusion

Nous avons défini les concepts de *mot*, de *langage* et l'ensemble des opérations applicables à ces objets. Bien que nous n'ayons couvert qu'une partie des opérations possibles, nous vous encourageons à consulter un cours de théorie des langages pour obtenir des informations plus détaillées.

II Les expressions rationnelles

Dans cette section, nous parlerons d'expressions régulières (ER). Nous allons nous concentrer sur un type bien particulier d'expressions régulières qui ne seront pas les expressions régulières que nous pouvons voir plus quotidiennement dans le domaine de l'informatique, les expressions régulières *UNIX*. Mais plutôt une version plus simple de celles-ci.

1 Définition

Nous allons noter une expression régulière $E \in Exp(\Sigma)$, c'est-à-dire une expression régulière où les symboles sont inclus dans l'ensemble Σ et où $Exp(\Sigma)$ représente l'ensemble des expressions sur Σ. Cette expression reconnait un langage qu'on pourra appeler $L(E)$. Nous pouvons définir une expression régulière récursivement de cette manière :

$$E = \varepsilon \tag{1}$$
$$E = a \tag{2}$$
$$E = F + G \tag{3}$$
$$E = F \cdot G \tag{4}$$
$$E = F^* \tag{5}$$
$$E = (F) \tag{6}$$
$$\text{avec } (E, F, G) \in (Exp(\Sigma))^3,\ a \in \Sigma$$

On notera que $*$ est prioritaire sur $\cdot$ qui est lui-même prioritaire sur $+$ et qu'ils sont tous deux associatifs à gauche. On comprend donc pourquoi l'équation (6) existe, elle est là pour des raisons de priorité. Il est alors évident de calculer les diverses fonctions sur celle-ci, c'est pour cela qu'on ne précisera pas son calcul. On peut définir chaque équation comme ceci :

- $E = \varepsilon$ **(epsilon) :** Représente le mot vide, de ce fait un mot de longueur zéro. Il peut être parfois représenté par « $ ».
- $E = a$ **:** Représente un symbole présent dans l'ensemble Σ
- $E = F + G$ **:** Représente l'union des deux expressions régulières F et G. Par abus de langage, on peut aussi dire F « ou » G pour représenter cette union.
- $E = F \cdot G$ **:** Représente la concaténation deux des deux expressions régulières F et G.

- $E = F^*$ **:** Représente la répétition infinie de F, cette répétition incluant, puissance 0 et donc le mot vide.

Pour calculer le langage que dénote l'expression régulière, on peut le calculer récursivement de cette manière :

$$L(\varepsilon) = \{\varepsilon\}$$
$$L(a) = \{a\}$$
$$L(F + G) = L(F) \cup L(G)$$
$$L(F \cdot G) = L(F) \cdot L(G)$$
$$L(F^*) = (L(F))^*$$
$$\text{Avec } a \in \Sigma \text{ et } (F, G) \in (Exp(\Sigma))^2$$

Example 2.0.1. On comprendra ainsi que l'expression $E = a + c \cdot d$ avec $E \in Exp(\Sigma)$ et $\Sigma = \{a, b, c, d\}$, dénote le langage $L(E) = \{a, cd\}$. Car on peut représenter E comme ceci :

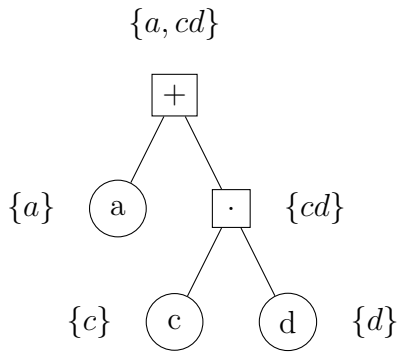

FIGURE 1 – Représentation de l'expression régulière à l'aide d'un arbre syntaxique

Comme on peut voir sur la Figure 1 grâce à cette représentation, on peut calculer simplement le langage reconnu par l'expression régulière (ici représenté par les ensembles à côté de chaque arbre).

Example 2.0.2. On comprendra aussi que l'expression $E' = \varepsilon + b^* \cdot a$, dénote le langage $L(E') = \{\varepsilon, b^* \cdot a\}$.

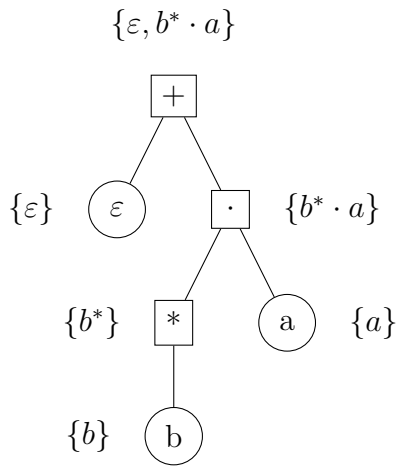

FIGURE 2 – Représentation de l'expression régulière à l'aide d'un arbre syntaxique

2 Fonction sur les *ER*

Une des fonctions les plus importantes sur les expressions régulières est $flnf$ permettant de calculer $FLNF(Exp(\Sigma))$ qui est un tuple défini comme ceci :

$$FLNF(Exp(\Sigma)) = (F, L, \Theta, \delta)$$

- $F \subseteq \Sigma$: Ensemble des premiers symboles de l'expression régulière
- $L \subseteq \Sigma$: Ensemble des derniers symboles de l'expression régulière
- $\Theta = \begin{cases} \varepsilon, & \text{si } \varepsilon \in L(E) \\ \varnothing & \text{sinon} \end{cases}$
- $\delta : \Sigma \to S$ avec $S \subseteq \Sigma$.
  Fonction renvoyant les symboles suivant du symbole passer en argument.

  La fonction $flnf$ a donc comme signature :

$$flnf : Exp(\Sigma) \to FLNF(Exp(\Sigma))$$

Et peut-être calculée de cette manière, pour $a \in \Sigma$ et $(E, G) \in (Exp(\Sigma))^2$:

$$flnf(\varepsilon) = (\varnothing, \varnothing, \varepsilon, \delta) \mid \delta(a) = \varnothing, a \in \Sigma$$

$$flnf(a) = (\{a\}, \{a\}, \varnothing, \delta) \mid \delta(a) = \varnothing, a \in \Sigma$$

$$flnf(E+G) = (F \cup F', L \cup L', \Theta \cup \Theta', \delta'') \text{ avec}$$
$$\delta''(a) = \delta(a) \cup \delta'(a) \mid \forall a \in \Sigma$$
$$(F, L, \Theta, \delta) = flnf(E) \wedge (F', L', \Theta', \delta') = flnf(G)$$

$$flnf(E \cdot G) = (F'', L'', \Theta \cap \Theta', \delta'') \text{ avec}$$
$$F'' = F \cup F' \cdot \Theta$$
$$L'' = L' \cup L \cdot \Theta'$$
$$\delta''(a) = \begin{cases} \delta(a) \cup \delta'(a) \cup F', & \text{si } a \in L \\ \delta(a) \cup \delta'(a) & \text{sinon} \end{cases} \mid \forall a \in \Sigma$$
$$(F, L, \Theta, \delta) = flnf(E) \wedge (F', L', \Theta', \delta') = flnf(G)$$

$$flnf(E^*) = (F, L, \varepsilon, \delta') \text{ avec}$$
$$\delta'(a) = \begin{cases} \delta(a) \cup F, & \text{si } a \in L \\ \delta(a) & \text{sinon} \end{cases} \mid \forall a \in \Sigma$$
$$(F, L, \Theta, \delta) = flnf(E)$$

Example 2.0.3. Prenons par exemple l'expression régulière suivante $E = a \cdot b + c \cdot d$, avec $E \in Exp(\Sigma)$ et $\Sigma = \{a, b, c, d\}$. Toujours à l'aide d'un arbre syntaxique, on peut calculer ce que $flnf(E)$ donnerait.

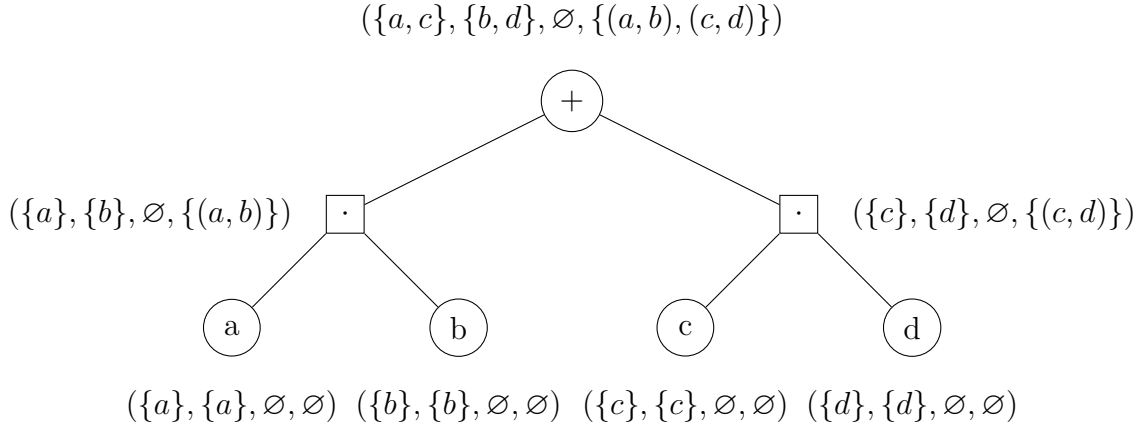

FIGURE 3 – Représentation de l'expression régulière à l'aide d'un arbre syntaxique. Pour des raisons de compréhension δ est représenté à l'aide d'un ensemble de couple.

Il advient que $flnf(E) = \{\{a, c\}, \{b, d\}, \varnothing, \delta\}$ avec δ qui est défini comme ceci :

$$\delta(a) = \{b\}$$
$$\delta(b) = \varnothing$$
$$\delta(c) = \{d\}$$
$$\delta(d) = \varnothing$$

Example 2.0.4. Un autre exemple pourrait être $E' = (a + b) \cdot c^*$, avec cet exemple, on voit l'utilité de la parenthèse, car sans elle la concaténation aurait été sur $b \cdot c^*$ et comme dit précédemment (1) son calcul reste le même que si c'était une union.

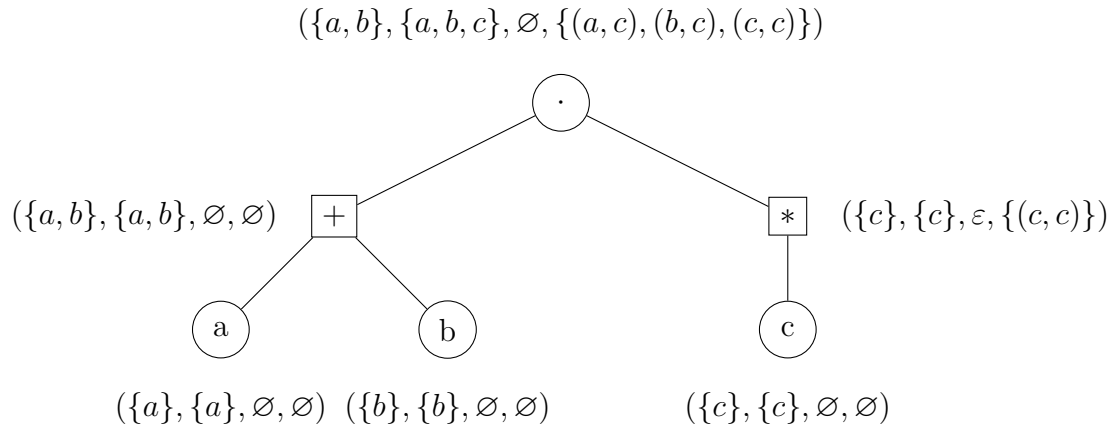

FIGURE 4 – Représentation de l'expression régulière à l'aide d'un arbre syntaxique. Pour des raisons de compréhension δ est représenté à l'aide d'un ensemble de couple.

Ce qui fait que $flnf(E') = (\{a,b\},\{a,b,c\},\varnothing,\delta')$ avec δ' qui est défini comme décrit après :

$$\delta'(a) = \{c\}$$
$$\delta'(b) = \{c\}$$
$$\delta'(c) = \{c\}$$
$$\delta'(d) = \varnothing$$

Une autre fonction qui s'applique sur les expressions régulières est *linearization* ; (elle peut paraitre inutile, mais) elle nous servira dans la Section IV. Sa signature est :

$$linearization : Exp(\Sigma) \to Exp(\Sigma \times \mathbb{N})$$

Elle peut être définie de cette manière, pour $a \in \Sigma$ et $(E, F) \in (Exp(\Sigma))^2$:

$$linearization(E) = \pi_2(linearization\_aux(E, 1)) \quad \text{avec}$$

Avec π_n la fonction de projection sur les tuples et *linearization_aux* défini récursivement comme ceci :

$$\begin{gathered}
linearization_aux(\varepsilon, n) = (\varepsilon, n) \\
linearization_aux(a, n) = ((a, n), n + 1) \\
linearization_aux(E + F, n) = (E' + F', n'') \quad \text{avec} \\
(E', n') \leftarrow linearization_aux(E, n) \\
(F', n'') \leftarrow linearization_aux(F, n') \\
linearization_aux(E \cdot F, n) = (E' \cdot F', n'') \quad \text{avec} \\
(E', n') \leftarrow linearization_aux(E, n) \\
(F', n'') \leftarrow linearization_aux(F, n') \\
linearization_aux(E^*, n) = (E'^*, n') \quad \text{avec} \\
(E', n') \leftarrow linearization_aux(E, n)
\end{gathered}$$

Avec cette définition, on peut voir que tous les symboles sont associés à un unique entier. Ce qui fait que l'expression régulière résultante ne contient que des symboles uniques. Et que de ce fait si deux couples partages le même entier cela implique qu'ils ont même valeur de symbole.

Example 2.0.5. Si on reprend cette expression régulière $E = \varepsilon + b^* \cdot b$, avec $E \in Exp(\Sigma)$ et $\Sigma = \{a, b, c, d\}$.

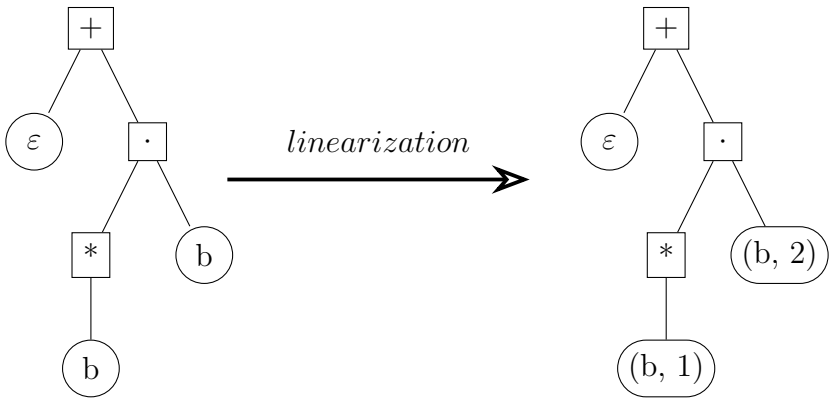

FIGURE 5 – Représentation à l'aide d'un arbre syntaxique de l'expression régulière une fois après avoir fait appel à *linearization* sur elle.

3 Conclusion

On saisit aisément que ces expressions ont beau être simples (peu d'opération comparé aux expressions régulières d'*UNIX*). Elles n'en sont rien pas complètes, on peut voir qu'elles permettent de décrire des langages très complexes et en quantité infinie. En revanche, il est difficile de savoir si un mot est reconnu par une expression régulière simplement. Par exemple est-ce-que le mot *eipipipipipip* est reconnu par cette expression $((((o \cdot \varepsilon) + (\varepsilon \cdot e)) + ((g \cdot \varepsilon) \cdot \varepsilon^*)) \cdot ((\varepsilon \cdot i) \cdot (p + \varepsilon))^*)$? La réponse est oui. C'est pour cela qu'il serait peut-être intéressant d'utiliser une autre structure de donnée pour reconnaitre des mots, comme les automates que nous allons voir maintenant.

III Les automates

Dans cette partie, nous parlerons des automates et plus particulièrement, nous allons parler des automates sans ε transition (de célèbre automate utilise des ε-transitions, comme ceux de *Thompson*, qui sont utilisés par nos ordinateurs). Pour autant les automates que nous verrons sont tout aussi bons que ceux avec ε transition.

1 Définition

Comme dit précédemment un automate est un objet mathématique reconnaissant un langage, on notera $M \in AFN(\Sigma, \eta)$ l'automate qui a pour transition des valeurs dans Σ, des valeurs « d'état » dans η. Et $AFN(\Sigma, \eta)$ l'ensemble des automates fini non déterministe de valeur de transition dans Σ et de valeur d'état dans η. On écrira donc $L(M)$ pour désigner le langage qu'il reconnait. Un automate est un tuple qu'on peut écrire de cette forme $M = (Q, I, F, \delta)$ avec :

$$
\begin{aligned}
Q &\subseteq \eta && \text{L'ensemble des états qui constitue l'automate} \\
I &\subseteq Q && \text{L'ensemble des états initiaux} \\
F &\subseteq Q && \text{L'ensemble des états finaux} \\
\delta &: Q \times \Sigma \to 2^Q && \text{La fonction de transition}
\end{aligned}
$$

Un automate peut se représenter à l'aide d'un graphe orienté, valué, particulier, Par exemple si on veut représenter $M = (\{q_1, q_2, q_3, q_4, q_5\}, \{q_1\}, \{q_2, q_3\}, \delta)$ avec $M \in AFN(\Sigma, \eta)$, $\Sigma = \{0, 1\}$, $\eta = \{q_1, q_2, q_3, q_4, q_5\}$ et δ défini comme ceci :

$$
\begin{aligned}
\delta(q_1, 0) &= \{q_2, q_4\} & \qquad \delta(q_3, 1) &= \{q_4\} \\
\delta(q_1, 1) &= \varnothing & \qquad \delta(q_4, 0) &= \{q_5\} \\
\delta(q_2, 0) &= \varnothing & \qquad \delta(q_4, 1) &= \{q_3\} \\
\delta(q_2, 1) &= \varnothing & \qquad \delta(q_5, 0) &= \{q_4\} \\
\delta(q_3, 0) &= \{q_3\} & \qquad \delta(q_5, 1) &= \{q_5\}
\end{aligned}
$$

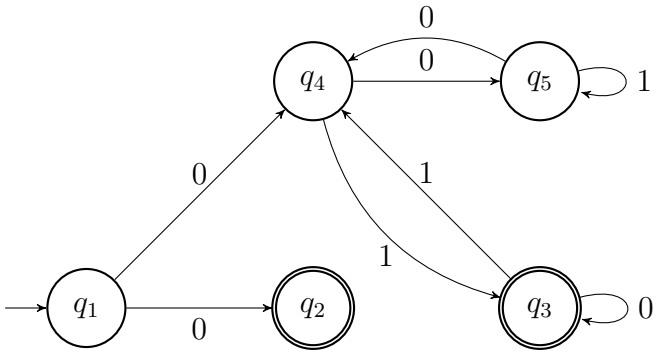

FIGURE 6 – Exemple de représentation graphique d'un automate.

Dans la Figure 6, on peut voir que les états initiaux (ici que q_1) ont une petite flèche qui pointe sur lui et que les états finaux ont un double contour. Et que les transitions sont symbolisées par des flèches entre les états et que ces flèches sont labellisées.

On parlera de l'inverse de l'automate M noté $\overleftarrow{M}$ qui peut être défini de cette façon :

$$\overleftarrow{M} = (Q, F, I, \delta') \quad \text{avec}$$
$$M = (Q, I, F, \delta)$$
$$\forall (p, q) \in Q^2 \mid q \in \delta(p, a) \Rightarrow p \in \delta'(q, a)$$

Donc si on veut représenter l'inverse de l'automate représenté dans la Figure 6, ça nous donnerait ceci :

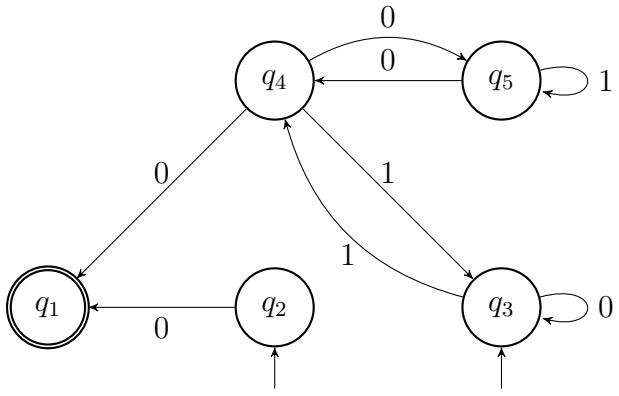

FIGURE 7 – Exemple de représentation graphique de l'inverse de l'automate de la Figure 6.

On voit bien que l'apparence de l'automate ne change pas les transitions sont juste inversées et les états initiaux sont devenus finaux et inversement.

On peut aussi étendre la fonction de transition δ de manière qu'elle ait comme signature :

$$\delta : Q \times \Sigma^* \to 2^Q$$

En la définissant récursivement de telle sorte :

$$\begin{aligned} \delta(q, \varepsilon) &= \{q\} \\ \delta(q, a \cdot w) &= \bigcup_{q' \in \delta(q,a)} \delta(q', w) \quad \text{avec } a \in \Sigma \end{aligned}$$

Example 3.0.1. Voici donc quelques exemples de ce que ça nous donnerait si on reprend l'automate utilisé pour la représentation graphique (Figure 6) :

$$\begin{gathered} \delta_w(q_1, 00) = \varnothing \\ \delta_w(q_1, 11) = \{q_3\} \\ \delta_w(q_1, \varepsilon) = \{q_1\} \\ \delta_w(q_1, 10 \cdot 1^*) = \{q_5\} \end{gathered}$$

Définition 3.1. Un automate est dit *standard* à l'instant où il ne possède qu'un seul état initial non ré-entrant, aussi défini comme ceci :

$$\begin{gathered} M = (Q, \{i\}, F, \delta) \quad \text{avec} \\ \forall p \in Q, \forall a \in \Sigma \mid i \notin \delta(p, a) \\ M \in AFN(\Sigma, \eta) \end{gathered}$$

Définition 3.2. Un automate est *homogène* quand pour tous les états les transition allant vers cet état ont même valeur. En d'autre terme quand il respecte cette propriété :

$$\begin{gathered} M = (Q, I, F, \delta) \quad \text{avec} \\ \forall (p, q, r) \in Q^3 \mid \exists (a, b) \in \Sigma^2 \wedge q \in \delta(p, a) \wedge q \in \delta(r, b) \Longrightarrow a = b \\ M \in AFN(\Sigma, \eta) \end{gathered}$$

Définition 3.3. Un automate est qualifié d'*accessible* lorsqu'en partant des initiaux, on peut arriver sur tous les états qui le composent. C'est-à-dire qu'il valide cette condition :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$\forall p \in Q, \exists w \in \Sigma^* \mid p \in \bigcup_{i \in I} \delta(i, w)$$
$$M \in AFN(\Sigma, \eta)$$

Lemme 3.4. Un automate est considéré comme *coaccessible* au moment où l'inverse de cette automate est accessible. Ceci veut dire qu'il atteste de cette particularité :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$accessible(\overleftarrow{M})$$
$$M \in AFN(\Sigma, \eta)$$

Définition 3.5. Un automate est dit *déterministe* quand tous ses états vont au maximum à un état par symbole. Autrement dit qu'il valide cette propriété :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$|I| = 1 \wedge \forall q \in Q, \forall a \in \Sigma \mid |\delta(q, a)| \leq 1$$
$$M \in AFN(\Sigma, \eta)$$

On parlera de *déterministe complet* lorsque tous ses états vont sur un état par symbole. C'est-à-dire qu'il respecte cette condition :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$|I| = 1 \wedge \forall q \in Q, \forall a \in \Sigma \mid |\delta(q, a)| = 1$$
$$M \in AFN(\Sigma, \eta)$$

Lemme 3.6. Un automate est qualifié de *hamac* à l'instant où il est standard, accessible et coaccessible. Ceci veut dire qu'il peut être décrit comme ceci :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$standard(M) \wedge accessible(M) \wedge coaccessible(M)$$
$$M \in AFN(\Sigma, \eta)$$

Donc l'automate représenté sur la Figure 6 est standard, non homogène, accessible et coaccessible. Car il possède bien un unique état initial (q_1), mais q_3, q_4 et q_5 ne respecte pas la propriété pour être homogène, parce qu'ils ont des transitions allant vers eux avec

des valeurs différentes. De plus tous ses états sont par accessible depuis l'état initial et son inverse est, lui-même aussi, accessible. Et bien sûr il n'est pas déterministe.

Les automates pouvant être représentés à l'aide de graphe, on peut étendre les propriétés sur les graphes aux automates. Par exemple, on pourra parler des composantes fortement connexes d'un automate. Autrement dit en partant de n'importe quel état, on peut arriver à tous les autres états. Ainsi ça veut dire qu'un automate fortement connexe vérifierait ceci :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$\forall (p, q) \in Q^2, \exists w \in \Sigma^* \mid q \in \delta_w(p, w)$$
$$M \in AFN(\Sigma, \eta)$$

2 Fonction sur les automates

Une des fonctions la plus importante sur les automates est *accept* qui test si le mot est reconnu par l'automate. C'est-à-dire que si on prend le chemin décrit par le mot donner en argument, on arrive sur un ou plusieurs états finaux. Elle a alors pour signature :

$$accept : AFN(\Sigma, \eta) \times \Sigma^* \to \mathbb{B}$$

Elle peut être définie simplement comme ceci :

$$accept(M, w) = \exists q \in \bigcup_{p \in I} \delta_w(p, w) \mid q \in F$$

3 Conclusion

Comme nous venons de voir les automates sont des outils puissants pour reconnaitre des mots d'un langage. L'un de leur plus grande force est leur simplicité, toutes les opérations sur les automates peuvent donc être automatisé. Ce qui fait que cet objet est très intéressant dans le monde de l'*informatique*. En revanche l'un de ses points faibles est sa représentation, il est difficile de représenter des très gros automates contrairement aux expressions régulières. Il serait alors intéressant de pouvoir convertir une expression régulière en automate. C'est ce que nous allons voir dans la prochaine section.

IV Les automates de Glushkov

Le terme « automate de Glushkov » est un abus de langage, faisant référence aux automates que l'algorithme de transformation d'expression régulière en automate, appelé algorithme de Glushkov produit. Son nom vient de l'informaticien soviétique *Victor Glushkov* qui est son créateur.

1 Définition

Nous appliquerons cet algorithme à l'aide la fonction *glushkov* qui a donc pour signature :

$$glushkov : Exp(\Sigma) \to AFN(\Sigma, \mathbb{N})$$

Et a ainsi, on peut définir cette fonction de cette façon :

$$\begin{aligned} glushkov(E) &= (Q, \{0\}, F, \delta) \quad \text{avec} \\ Q &\leftarrow \{n \mid n \in \mathbb{N} \wedge 0 \leq n < m\} \\ F &\leftarrow \{n \mid (a, n) \in Last\} \cup (\{0\} \cdot Null) \\ \forall q \in \delta(p, a) \mid & \begin{cases} (a, q) \in First, & \text{si } p = 0 \\ (a, q) \in Follow((b, p)) & \text{sinon} \end{cases} \\ (E', m) &\leftarrow linearization(E) \\ (First, Last, Null, Follow) &\leftarrow flnl(E') \end{aligned}$$

Example 4.0.1. Vu qu'un dessin vaut toujours mieux que mile mots, voici un exemple de l'automate résultant de la transformation de cette expression $E = (a + b) \cdot a^* \cdot b^* \cdot (a + b)^*$.

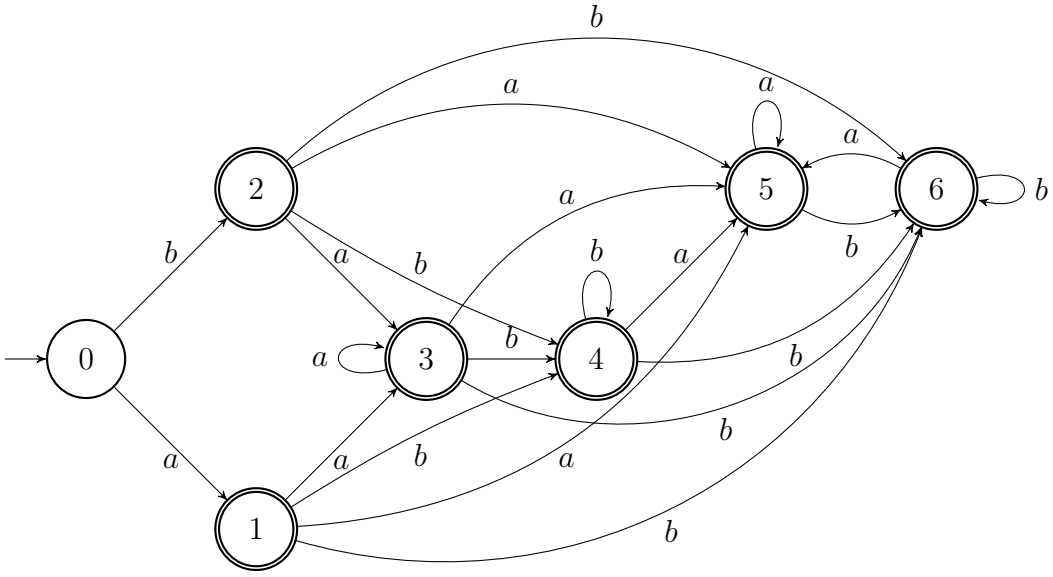

FIGURE 8 – Exemple de représentation graphique de l'automate résultant de $glushkov(E)$.

Comme on peut voir sur la Figure 8 les automates produits sont bien souvent gros et peuvent être difficile à comprendre, mais une machine peut gérer ça très simplement. Outre sa taille, on peut remarquer qu'il y a des propriétés intéressant sur cette automate. C'est ce que l'on va étudier maintenant.

2 Propriétés :

Nous verrons ici plusieurs propriétés sur les automates de Glushkov, mais nous n'en ferons pas la preuve, nous en donnerons une justification, mais pas une réelle preuve (preuve disponible dans ce document [CZ00]).

- La première propriété plutôt évidente, est que les automates de Glushkov sont *standards*, car par constructions, il ne peut avoir qu'un seul état initial (0).

- La deuxième est que l'automate à $n + 1$ avec n le nombre de symboles de l'expression régulière. Le $+1$ vient dû fait que nous ajoutons un état 0 qui a des transitions vers les $First$.

- La troisième propriété un peu moins flagrante est que les automates de Glushkov sont accessibles et coaccessibles. C'est dû au fait que chaque symbole dans l'expression régulière est accessible et coaccessible et que cette propriété ne se perd pas lors de la transformation.

- La dernière propriété est que l'automate de Glushkov est homogène. Cela résulte de sa construction, car pour qu'un état aille sur un autre état, il faut qu'il ait dans ses *Follow* (a, n) avec a le symbole de la transition et n la valeur de l'état. Et étant donné que pour chaque couple (b, m) il ne peut n'avoir que ce couple avec comme seconde valeur m alors la transition vers cet état sera toujours la même.

3 Conclusion

L'algorithme de Glushkov est très puissants, car permet de convertir une expression régulière en automate ce qui fait qu'on gagne les avantages des deux structures. Avec les expressions régulières, on peut simplement décrire un langage et avec les automates, on peut simplement savoir si un mot est reconnu. Il est très utilisé en *informatique*, parce que pour les humains, il est plus simple de décrire un langage avec une expression régulière. Et les machine comprennent très facilement les automates. Ce qui fait qu'il est possible de faire des *programmes informatiques* qui reconnait un langage et exécute des tâches à chaque mot.

V Conclusion

Ce document a fourni une définition formelle des concepts clefs utilisés dans la théorie des langages et les automates. Nous avons d'abord introduit les notions de mots, de langages et les opérations fondamentales qui leur sont associées. Ensuite, nous avons exploré les expressions régulières, leurs définitions et les fonctions qui peuvent être appliquées sur elles. Par la suite, nous avons étudié les automates, en particulier ceux sans transitions ε, et les fonctions qui leur sont appliquées. Enfin, nous avons abordé les automates de Glushkov, décrivant leur construction et leurs propriétés.

Bien que nous n'ayons couvert qu'une partie des concepts et des opérations possibles, cette introduction vise à fournir une base solide pour comprendre et utiliser ces outils puissants. Pour approfondir vos connaissances, nous vous encourageons à consulter des ressources supplémentaires en théorie des langages et des automates.

Bibliographie

[Har78] Michael A. Harrison. *Introduction to formal language theory*. English. Addison-Wesley series in computer science. Reading, Mass. : Addison-Wesley Pub. Co., 1978. isbn : 0201029553 ; 9780201029550.

[Aut94] Jean-Michel Autebert. *Théorie des langages et des automates*. French. Manuels informatiques Masson. Paris : Masson, 1994. isbn : 2225840016 ; 9782225840012.

[CZ00] Pascal Caron et Djelloul Ziadi. « Characterization of Glushkov automata ». In : *Theor. Comput. Sci.* 233.1-2 (2000), p. 75-90.

[HMU07] 1939- Hopcroft John E., Rajeev Motwani et 1942- Ullman Jeffrey D. *Introduction to automata theory, languages, and computation*. English. 3rd ed. Boston : Pearson/Addison Wesley, 2007. isbn : 0321455363 ; 9780321455369 ; 0321462254 ; 9780321462251 ; 0321455371 ; 9780321455376 ; 0321476174 ; 9780321476173.

[Car23] Pascal Caron. « Cours de théorie des langages ». Non publié. Document interne à l'Université de Rouen. Déc. 2023.